Building my own Border Router & Wireguard Love Story

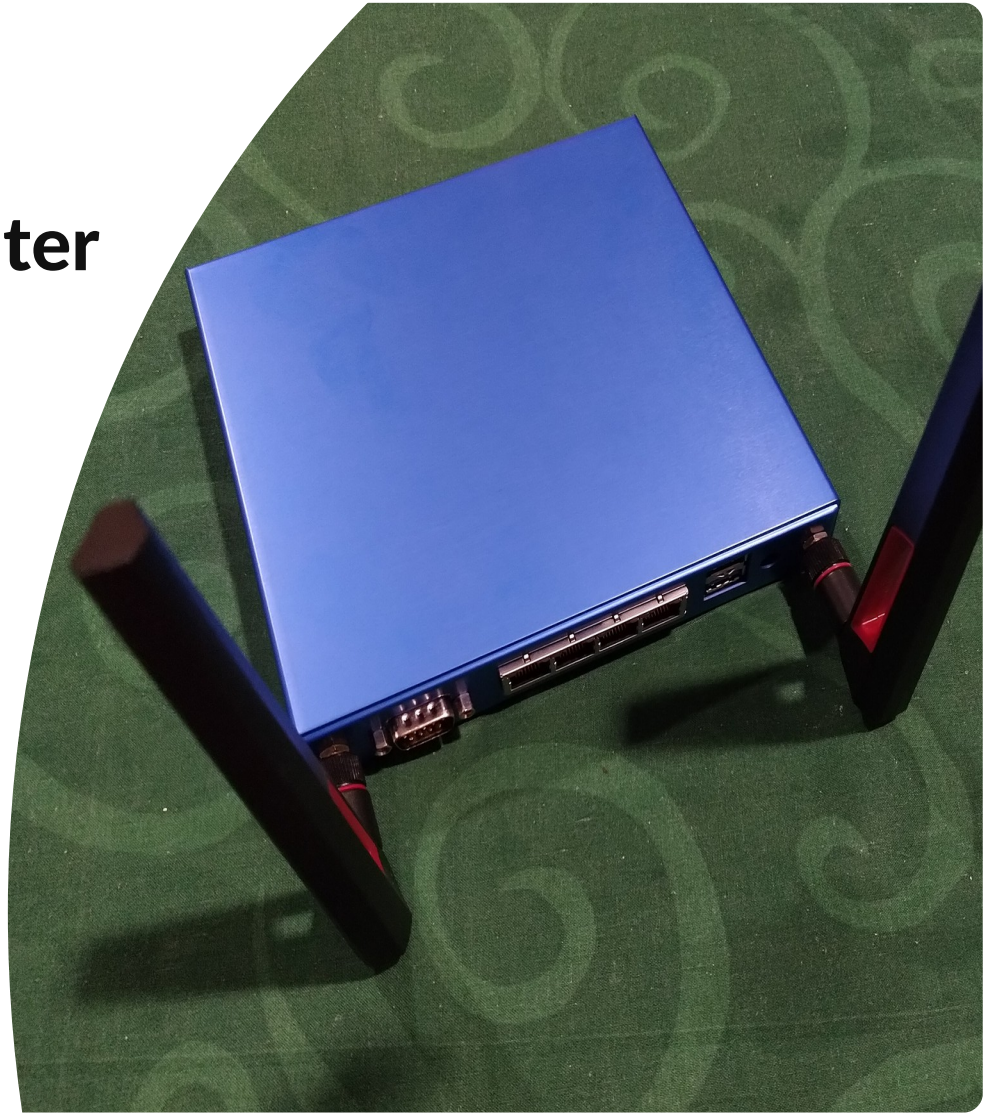
Arjen Lentz

CISO, Catalyst IT Australia

arjenlentz@catalyst-au.net

catalyst

expert open source solutions



What are we trying to achieve?

- Learning experience
  - Exploring it all yourself adds perspective
  - Good testbed
- Improved yet maintainable security
  - Few tricks cover all scenarios
  - Does one trick catch lots? Great!
  - Looking which bits work, and overall effect
  - No single “this is right and that is wrong”

Hardware

pcengines.ch *by Pascal Dornier & co, since 2002*

System board: APU4d4 (€ 99) *schematics available*

- AMD Embedded G series GX-412TC
- 1GHz quad 64-bit Jaguar core *(conductive cooling)*
- BIOS based on Coreboot *(source code on GitHub)*
- 4GB DDR3-1333 DRAM
- 4x gigabit Ethernet (Intel i211AT)
- DB9 serial, 2xUSB3.0, 2xUSB2.0, SD, SATA, GPIO, I2C
- 3x miniPCI express (mSATA, WiFi, SIM)
  - I use 120GB mSATA, WLE600VX WiFi

Software

- Debian 10 *(could be Ubuntu with tweakage)*
  - Install from USB + serial console
- [LUKS,] (LVM + ext4) | BtrFS
- USBguard, *sorry no RKhunter/Suricata today*
- bridge-utils, vlan, pppd, hostapd (for wifi)
- iptables/ipset, iptables-persistent, DNSmasq
- OpenSSH, Mosh, screen, vim
- WireGuard *(needs repo in Deb10)*

Disk layout

I used LVM + multiple partitions, but doh... you can do all this in BtrFS with subvolumes!

Security: *mounts are either writeable by non-priv users, or executable, but never both*

- /
  - /boot nodev
  - /usr [ro,]nodev <-- *I tried read-only and it should be ok, but there are issues*
  - /var nodev
  - /home nodev,nosuid,noexec
  - /tmp nodev,nosuid,noexec
- Swap *(file or partition)*

/etc/network/interfaces

- **enp1s0** static
  - mtu 9000
- **wlp5s0** static
  - hostapd <conffile>
- **br0** static <-- building our on little switch here
  - *!! setting MTU on a bridge will make ifup fail*
  - bridge_ports enp2s0 enp3s0, bridge_stp off, bridge_fd 0, bridge_maxwait 5
  - pre-up /sbin/ip link set enp2s0 up; /sbin/ip link set enp3s0 up
- ...

/etc/network/interfaces ... my Internode NBN via HFC

- **vlan2** manual
  - vlan-raw-device enp4s0
  - pre-up /sbin/ip link set enp4s0 up
  - mtu 1508
- **enp4s0** manual
  - hwaddress <MAC address of my “official” router>
- **hfc** ppp <-- *actually ends up as interface ‘internode’ when up, through config*
  - pre-up /sbin/ip link set vlan2 up
  - provider hfc-internode

/etc/sysctl.d/99-z-yourstuff.conf

net.core

.netdev_max_backlog = 262144
.rmem_default = 31457280
.rmem_max = 67108864
.wmem_default = 31457280
.wmem_max = 67108864
.somaxconn = 65535
.optmem_max = 25165824

net.netfilter.nf_conntrack

_max = 10000000
_tcp_loose = 0 <-- *this one is a doozy*
_tcp_timeout_established = 1800
_tcp_timeout_close = 10
_tcp_timeout_close_wait = 10
_tcp_timeout_fin_wait = 20
_tcp_timeout_last_ack = 20
_tcp_timeout_syn_recv = 20
_tcp_timeout_syn_sent = 20
_tcp_timeout_time_wait = 10

/etc/sysctl.d/99-z-yourstuff.conf

net.ipv4.neigh
.default.gc_thresh1 = 4096
.default.gc_thresh2 = 8192
.default.gc_thresh3 = 16384
.default.gc_interval = 5
.default.gc_stale_time = 120

net.ipv4.conf
.default.rp_filter = 2 <-- *for VPNs*
.all.rp_filter = 2 ...
.lo.rp_filter = 0
.all.secure_redirects = 1
.all.log_martians = 1

net.ipv4.ip_forward = 1 <-- *Router*

/etc/sysctl.d/99-z-yourstuff.conf

net.ipv4
.ip_local_port_range = 1024 65000
.ip_no_pmtu_disc = 1

.route.flush = 1
.route.max_size = 8048576

.icmp_echo_ignore_broadcasts = 1
.icmp_ignore_bogus_error_responses = 1

.udp_mem = 65536 131072 262144
.udp_rmem_min = 16384
.udp_wmem_min = 16384

Daunted? Yea, I was too…

*The key point is, by default the Linux kernel networking foo is **not** particularly*

- *Tuned [well]*
- *Secure*
- *Resistant to floods and DDoS*

So one should tweak it "a bit". Ha.

Worst thing: try to find docs | understand

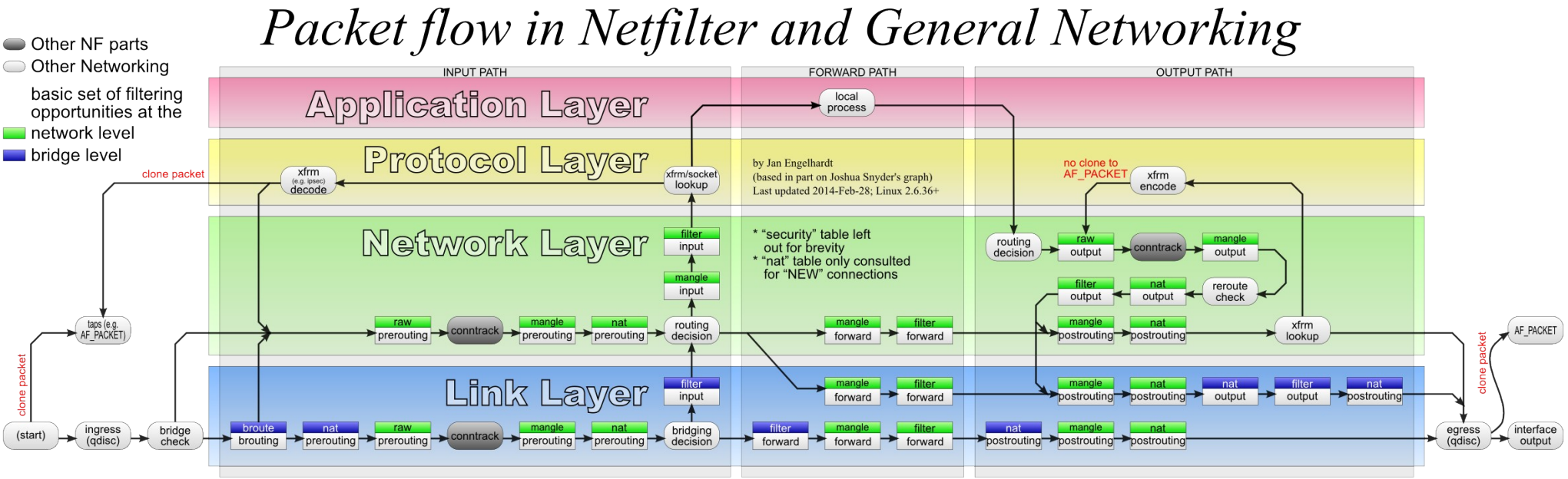
Take a deep breath, we're almost there…

/etc/sysctl.d/99-z-yourstuff.conf

net.ipv4
.tcp_slow_start_after_idle = 0
.tcp_congestion_control = htcp
.tcp_mem = 65536 131072 262144
.tcp_rmem = 4096 87380 33554432
.tcp_wmem = 4096 87380 33554432
.tcp_max_tw_buckets = 1440000
.tcp_tw_reuse = 1
.tcp_max_orphans = 400000
.tcp_window_scaling = 1
.tcp_rfc1337 = 1
.tcp_syncookies = 1

net.ipv4
.tcp_synack_retries = 1
.tcp_syn_retries = 2
.tcp_max_syn_backlog = 16384
.tcp_timestamps = 0 <-- *or 1, as you please*
.tcp_sack = 1
.tcp_fack = 1
.tcp_ecn = 2
.tcp_fin_timeout = 10
.tcp_keepalive_time = 600
.tcp_keepalive_intvl = 60
.tcp_keepalive_probes = 10
.tcp_no_metrics_save = 1

Now for our firewall... a quick overview. Let's see...



Aka: eek!

Actual lessons to guide our firewall bricklaying

So it's extensive. Yeah. Google, Cloudflare, etc run on Linux, too. It can do the job!

- Cloudflare says: "If you're going to drop a packet, do it as early as possible."

  1) Raw

  2) Mangle

  3) Filter

  4) NAT

- Arjen says: "The less noise I let in, the less noise I have to monitor."

  1) LOG what we want to be filtering. Patience! Am I happy with what it catches? (anything?)

  2) DROP those weird packets

  3) Regularly grep around in /var/log/syslog to see what we've caught

  4) Use Cloudflare's **mmwatch 'sudo iptables-save -c | egrep "^\[" | grep -v "LOGDROP "**

SYN PROXY or not?

*raw
-A PREROUTING -i *internode* -p tcp -m tcp -m multiport --dports 80,443,...
--syn -j CT --notrack

*filter
-A INPUT -i *internode* -p tcp -m state --state UNTRACKED,INVALID
-j SYNPROXY --sack-perm --timestamp --mss 1460 --wscale 9

I didn't get any joy out of this, so not using it right now. You, perhaps?

I mentioned logging before dropping

Logging everything in this rule target:

-A mPRE_LOGDROP -m limit -j LOG --log-prefix "iptables mPRE: " --log-level 5
-A mPRE_LOGDROP -j DROP

or if you want only a sample, add this:
    --limit 3/min --limit-burst 20

My naming scheme:

- *m for Mangle table*
- *PRE = PREROUTING, POST = POSTROUTING, FWD = FORWARD, etc*

PREROUTING in Mangle table

 # forget doing complex xmas packet detection, these do the job.
[455767:25557562] -A PREROUTING -m state --state INVALID -j mPRE_LOGDROP
[43891:7364504] -A PREROUTING -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,PSH,ACK,URG SYN -m state
--state NEW -j DROP
 # weirdly sized packets: ditch
[2556926:102280488] -A PREROUTING -i *internode* -p tcp -m conntrack --ctstate NEW -m tcpmss
! --mss 536:65535 -j mPRE_LOGDROP
 # no "RELATED", it's dangerous. We don't have FTP and we can deal with SIP separately
-A PREROUTING -m state --state ESTABLISHED -j ACCEPT
 # fragmented packets: ditch (interesting eh?) I don't actually see hits on this one anyway
-A PREROUTING -i *internode* -f -j DROP
 # More on this... coming up!
-A PREROUTING -i *internode* -j mBLOCK
 # Ensure outbound sanity
-A FORWARD -o internode -p tcp -m tcp --tcp-flags SYN,RST SYN -m tcpmss
--mss 1400:65495 -j TCPMSS --clamp-mss-to-pmtu

mBLOCK: RETURN = allow through

[133433:8797140] -A mBLOCK -m set --match-set inet.pingdom.list src -j RETURN
-A mBLOCK -s <friendlyip>/32 -j RETURN
-A mBLOCK -s <friendlynet>/24 -j RETURN

 # Yep, geo blocking. Reduce noise
[240499:17237381] -A mBLOCK -m set --match-set inet.au.zone src -j RETURN
[309810:72386300] -A mBLOCK -m set --match-set inet.nl.zone src -j RETURN

 # Would be nice to also have a list for Let's Encrypt... *they don't give one!*
 # We can probably get it down to a [few] AS network(s), through monitoring?
 # For now we have to allow the very noisy US block through when updating our certificates

 # Invalid network ranges of any kind
[18136:1015452] -A mBLOCK -m set --match-set inet.fullbogons.list src -j DROP

 # Swift byebye to everything else
[2970699:216113048] -A mBLOCK -j DROP

So ipset is our friend, feed it regularly

```
wget -O inet.pingdom.list https://my.pingdom.com/probes/ipv4
wget -O inet.fullbogons.list \
    https://www.team-cymru.org/Services/Bogons/fullbogons-ipv4.txt
wget -O inet.${COUNTRY}.zone \
    https://www.ipdeny.com/ipblocks/data/aggregated/${COUNTRY}-aggregated.zone

# hash:ip or hash:net, set timeout to 30 days, have counters to monitor in more detail
ipset -exist create inet.${COUNTRY}.zone hash:net family inet timeout 2592000 counters
while read line; do
    ipset -exist add $1 $line
done <$1
```

INPUT in Filter table (default=DROP)

-A INPUT -i lo -j ACCEPT
-A INPUT -i enp1s0 -s <localnet>/24 -j ACCEPT
-A INPUT -m state --state ESTABLISHED -j ACCEPT

 # Only accept SSH/Mosh from local network or via VPN
-A INPUT ! -i *internode* -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT ! -i *internode* -p udp -m multiport --dports 60000:61000 -j ACCEPT

 # Restrict/limit ping
-A INPUT -p icmp -m icmp --icmp-type 8/0 -m state --state NEW,RELATED -m limit --limit 50/sec -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 3/4 -m limit --limit 50/sec -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 11/0 -m limit --limit 50/sec -j ACCEPT
-A INPUT -p icmp -j DROP

 # Rough general ruleset for hosts: add specific ports to accept – flood tracking per individual IP
-A INPUT -p tcp -m connlimit --connlimit-above 111 --connlimit-mask 32 --connlimit-saddr -j REJECT --reject-with tcp-reset
-A INPUT -p tcp -m tcp --tcp-flags RST RST -m limit --limit 2/sec --limit-burst 2 -j ACCEPT
-A INPUT -p tcp -m tcp --tcp-flags RST RST -j IN_LOGDROP
-A INPUT -p tcp -m conntrack --ctstate NEW -m limit --limit 60/sec --limit-burst 20 -j ACCEPT
-A INPUT -p tcp -m conntrack --ctstate NEW -j IN_LOGDROP

FORWARD in Filter table (default=DROP)

-A FORWARD -i enp1s0 -s <localnet>/24 -j ACCEPT
-A FORWARD -s <friendly-ip>/32 -j ACCEPT

-A FORWARD -m state --state ESTABLISHED -j ACCEPT

-A FORWARD -p icmp -m icmp --icmp-type 8/0 -m state --state NEW,RELATED -m limit --limit 50/sec -j ACCEPT
-A FORWARD -p icmp -m icmp --icmp-type 3/4 -m limit --limit 50/sec -j ACCEPT
-A FORWARD -p icmp -m icmp --icmp-type 11/0 -m limit --limit 50/sec -j ACCEPT
-A FORWARD -p icmp -j DROP

 # More restrictive acceptance: only for specific ports
 # Limit floods per /24 subnet (that makes for a table of max 16 million entries, we can easily handle that)
-A FORWARD -i *internode* -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -m multiport --dports 80,443,465,587,993
    -m connlimit --connlimit-above 16 --connlimit-mask 24 --connlimit-saddr -j FWD_LOGDROP

-A FORWARD -d <bridged-ip>/32 -p tcp -m multiport --dports 80,443 -j ACCEPT
-A FORWARD -d <bridged-ip>/32 -p tcp -m multiport --dports 80,443,465,587,993 -j ACCEPT

DNSmasq: DHCP server, DNS cache, ad&bad filtering + more

/etc/dnsmasq.conf

```
# Use Cloudflare's DNS, with extra malware filtering
server=/#/1.1.1.2
cache-size=8192

addn-hosts=/etc/dnsmasq_ad_servers
addn-hosts=/etc/dnsmasq_bad_servers
```

We feed this wisdom with:

```
wget -O - "https://pgl.yoyo.org/as/serverlist.php?hostformat=nohtml" \
      | awk '{printf "0.0.0.0 %s\n", $1}' >/etc/dnsmasq_ad_servers

wget -O - "https://someonewhocares.org/hosts/zero/" \
      | grep "^0.0.0.0" >/etc/dnsmasq_bad_servers
```

iptables for DNSmasq (filter table)

 # Let through response-traffic for DNS and NTP
-A INPUT -i *internode* -p udp -m state --state ESTABLISHED -m udp
-m multiport --sports 53,123 -j ACCEPT

 # Allow local DNS requests
-A INPUT -i enp1s0 -s <localnet>/24 -i enp1s0 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i enp1s0 -s <localnet>/24 -i enp1s0 -p tcp -m tcp --dport 53 -j ACCEPT

-A INPUT -i br0 -s <bridge-ips>/29 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i br0 -s <bridge-ips>/29 -p tcp -m tcp --dport 53 -j ACCEPT

 # Allow local DHCP requests
-A INPUT -i enp1s0 -p udp -m udp --dport 67 -j ACCEPT

Could we filter potentially bad outbound IP traffic? Hmm...

 # /etc/rc.local
ipset -exist create inet.dnsmasq.list hash:ip family inet timeout 86400

 # /etc/dnsmasq.conf
 # This is fabulous: DNSmasq can add IPs from a DNS lookup to an ipset!
ipset=/#/inet.dnsmasq.list

 # In iptables we may need a few exceptions – yea we need to restrict the NTP better
 # Hint: if you're a gamer, you may need to monitor and add some more exceptions
-A FORWARD -o *internode* -p udp -m udp -m multiport --dports 123 -j ACCEPT

 # So any routed outbound traffic that's not in the DNSmasq ipset... drop!
 # Malware/C2/exfiltration often use direct IPs, DNS-over-TLS, DNS-over-HTTPS, etc.
-A FORWARD -o *internode* -m set ! --match-set inet.dnsmasq.list dst -j MASQ_LOGDROP

It's tiny (few thousand lines of code), secure, and pretty fast. Easy yet powerful. I love it.

By Jason Donenfeld

Kernel module via repo/DKMS, in 5.6 kernel, or Ubuntu 5.4 kernel

```
$ sudo lsmod | grep wireguard
wireguard                  94208  0
curve25519_x86_64          49152  1 wireguard
libcurve25519_generic      49152  2 curve25519_x86_64,wireguard
libchacha20poly1305        16384  1 wireguard
ip6_udp_tunnel             16384  1 wireguard
udp_tunnel                 16384  1 wireguard
libblake2s                 16384  1 wireguard
```

WireGuard: server setup - complete

wg genkey | tee server-private.sec | wg pubkey > server-public.key

/etc/wg0.conf

 # Of course, add iptables rules for wg0

[Interface]
Address = <server-private-addr>/24
Table = off
SaveConfig = true
ListenPort = 51820
FwMark = 0xca6c
PrivateKey = <content of server-private.sec>

 # My roaming laptop
[Peer]
PublicKey = <peer-pub-key>
AllowedIPs = <peer-pvt-addr>/32

 # VPN host in other country
[Peer]
PublicKey = <peer-pub-key>
AllowedIPs = <peer-pvt-addr>/32, 0.0.0.0/0
Endpoint = <peer-pub-addr>:51820

My roaming laptop, always on wireguard - complete

a)Ensure wg0 goes up (or even down/up) when any wired/wireless iface goes up

b)You can point resolv.conf at the nameserver on the VPN server (or run a local DNSmasq)

 # /etc/wireguard/wg0.conf
[Interface]
PrivateKey = <our-private-key>
Address = <our-pvt-addr>

[Peer]
PublicKey = <server-public-key>
Endpoint = <server-public-addr>:51820
AllowedIPs = 0.0.0.0/0

VPN host in other country - complete

[Interface]
Address = <our-pvt-addr>/32
SaveConfig = true
PostUp = iptables -t nat -A POSTROUTING -s <our-pvt-addr>/24 -o ens3 -j SNAT --to-source <our-public-addr>
PostDown = iptables -t nat -D POSTROUTING -s <our-pvt-addr>/24 -o ens3 -j SNAT --to-source <our-public-addr>
ListenPort = 51820
PrivateKey = <our-private-key>

 # This is the link back to our main VPN host
[Peer]
PublicKey = <peer-pub-key>
AllowedIPs = <peer-pvt-addr>/32
Endpoint = <peer-pub-addr>:51820

Geo-based VPN using policy-based routing: iptables

 # General rules for WireGuard – you can/should make this more strict all the way to zero-trust!
-A INPUT -i wg0 -j ACCEPT
-A FORWARD -i wg0 -j ACCEPT
-A FORWARD -o wg0 -j ACCEPT
-A OUTPUT -o wg0 -j ACCEPT

 # Policy-based routing
-A PREROUTING -j mPOLROUTE
-A FORWARD -j mPOLROUTE
-A OUTPUT -j mPOLROUTE

-A mPOLROUTE -d <peer-pub-addr>/32 -j RETURN
-A mPOLROUTE -m set --match-set <country-net-list> dst -j MARK --set-xmark 0xc21/0xffffffff
-A mPOLROUTE -j RETURN

Geo-based VPN using policy-based routing: routing

systemctl start wg-quick@wg0

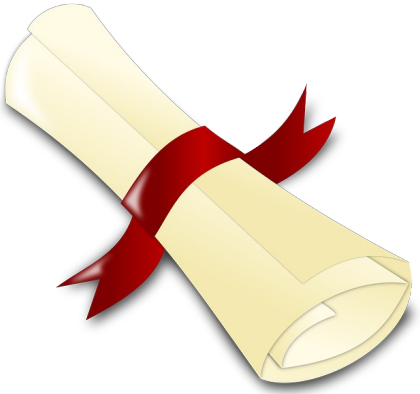
ip route add <server-pvt-addr> dev wg0

ip route add table wg.some.country default via <server-pvt-addr> dev wg0

ip rule add priority 1000 from all fwmark 3105 lookup wg.some.country

ip rule add table main suppress_prefixlength 0

ip route flush cache



Thank You

Open For Questions

arjenlentz (at) catalyst-au.net
arjen (at) lentz.com.au

Twitter @arjenlentz

github.com/arjenlentz/routerfoo
(populated soon, PRs welcome)

linkedin.com/in/arjenlentz

catalyst